Курсовой проект по курсу : Операционные системы

Выполнил студент группы М8О-201Б-21 Кварацхелия Александр.

Условие

Ознакомиться с сигналами операционной системы UNIX/LINUX, используя утилиту strace, проанализировать результаты, сопоставить их с кодом программы.

Метод решения

Использовать свободно распространяемую утилиту strace следующим образом: strace lab2

Код программы

zmq-utils.hpp

```
#ifndef ZMQ_UTILS_HPP
#define ZMQ_UTILS_HPP

#include <string>
#include <iostream>
#include <zmq.hpp>

namespace zus
{
    class Socket
    {
        public:
            Socket(const std::string &address, const std::string &socketType)

            void SendMessage(const std::string &requestStr);
            std::string RecieveMessage();
            std::string SendRequest(const std::string &requestStr);
            std::string SendResponse(const std::string &responseStr);

            ~Socket();

        private:
            std::string address;
            std::string socketType;
```

```
            zmq::context_t context;
            zmq::socket_t socket;

            struct Private;
    };
}


#endif
```

server-utils.hpp

```
#ifndef SERVER_UTILS_HPP
#define SERVER_UTILS_HPP

#include <string>

int Unpack(std::string &str);

#endif
```

utils.hpp

```
#ifndef UTILS_HPP
#define UTILS_HPP

#include <string>
#include <regex>
#include <set>

class RoomName
{
    public:
        RoomName();
        RoomName(std::string &name);

        std::string data;
        std::string formatted;

        ~RoomName();
    private:
        ;
};
```

```
std::istream &operator >> (std::istream &in,  RoomName &obj);
```

```
class Room
{
    public:
        RoomName name;
        std::string port;
        bool running{0};

        Room(std::string &line);
        Room(std::string &name, std::string &port);
        Room(std::string &name, std::string &port, bool running);
        ~Room();

    private:
};
```

```
class RoomsManager
{
    public:
        RoomsManager();

        bool Member(std::string &name);
        std::string GetPort(std::string &name);
        std::string AddRoom(std::string &name);
        void StartRoom(std::string &name);

        ~RoomsManager();
    private:
        std::set<Room> rooms;
};
```

```
#endif
```

zmq-utils.cpp

```
#include "zmq_utils.hpp"
#include "allias.hpp"
#include <exception>
#include <zmq.hpp>

struct zus::Socket::Private
{
    static void SendMessage(zus::Socket &self, const std::string &requestStr)
    {
        try
        {
            zmq::message_t response(requestStr.data(), requestStr.length());
            zmq::send_result_t responseStatus = self.socket.send(response, zm
        }
        catch(std::exception &exc){
            std::cerr << "[socket:␣" << self.address << "]␣" << exc.what() <<
        }
    }

    static std::string RecieveMessage(zus::Socket &self)
    {
        std::string result;

        try
        {
            zmq::message_t request;
            zmq::recv_result_t requestStatus = self.socket.recv(request, zmq:
            result = request.to_string();
        }
        catch(std::exception &exc){
            std::cerr << exc.what() << std::endl;
        }

        return result;
    }
};
```

```
zus::Socket::Socket(const std::string &address, const std::string &socketType
{
    this->socketType = socketType;
    this->address = address;

    if (!socketType.compare(utl::SERVER)){
        this->socket = zmq::socket_t(this->context, zmq::socket_type::rep);
    }

    if (!socketType.compare(utl::CLIENT)){
        this->socket = zmq::socket_t(this->context, zmq::socket_type::req);
    }

    try
    {
        if (!this->socketType.compare(utl::SERVER)){
            this->socket.bind(address);
        }

        if (!this->socketType.compare(utl::CLIENT)){
            this->socket.connect(address);
        }
    }
    catch (std::exception &exc){
        //std::cerr << sym::RED << "[Error:]" << sym::RESET \
        //    << " socket: " << address << ": " \
        //    << exc.what() << std::endl;
    }
}

void zus::Socket::SendMessage(const std::string &requestStr)
{
    try
    {
        zmq::message_t response(requestStr.data(), requestStr.length());
        zmq::send_result_t responseStatus = this->socket.send(response, zmq::
    }
    catch(std::exception &exc){
        std::cerr << "[socket: msg: " << ", " << this->address << "] " << exc
    }
```

```
}

std::string zus::Socket::RecieveMessage()
{
    std::string result;

    try
    {
        zmq::message_t request;
        zmq::recv_result_t requestStatus = this->socket.recv(request, zmq::re
        result = request.to_string();
    }
    catch(std::exception &exc){
        std::cerr << exc.what() << std::endl;
    }

    return result;
}

std::string zus::Socket::SendRequest(const std::string &requestStr)
{
    //Private::SendMessage(*this, requestStr);
    //return Private::RecieveMessage(*this);
    SendMessage(requestStr);
    std::string responseStr = RecieveMessage();

    return responseStr;
}

std::string zus::Socket::SendResponse(const std::string &responseStr)
{
    std::string requestStr = RecieveMessage();
    SendMessage(responseStr);

    return responseStr;
}

zus::Socket::~Socket()
{

    if (!this->socketType.compare(utl::CLIENT))
    {
```

```
        std::string request = utl::TERMINATOR;
        SendRequest(request);
        this->socket.disconnect(this->address);
    }

    this->socket.close();
    this->context.close();
}
```

client.cpp

```
#include "zmq_utils.hpp"
#include "utils.hpp"
#include "allias.hpp"

#include <cstdlib>
#include <iostream>
#include <iterator>
#include <string>
#include <regex>
#include <zmq.hpp>

std::string WithId(std::string &id, std::string &str){
    return std::string(id + "␣" + str);
}

int main(int argc, char* argv[])
{
    std::string port = argv[1];
    std::string id = argv[2];

    std::string address = "tcp://localhost:" + port;
    zus::Socket socket(address, utl::CLIENT);

    ClientCommand cmd(id);

    std::string input, response, request = cmd.INIT_CLIENT;

    std::cout << "Log␣in␣->␣";
    std::cin >> input;
    request += "␣" + input + "␣";

    response = socket.SendRequest(request);
```

```
    request = id + " ";

    std::cout << "\nCREATE - create room,\n CONNECT - connect to the room.\n-
    std::cin >> input;
    request += input;

    std::cout << "\nEnter lobby name ->";
    std::cin >> input;
    request += " " + input;

    std::cout << request << '\n';
    response = socket.SendRequest(request);

    std::cout << "\nCommands:\n0 - start game;\n" \
        << "1 <number> - choose <number>\n" \
        << "2 <user> - print stat of <user>\n";

    while (std::getline(std::cin, input))
    {
        if (input.length())
        {
            if (input.compare(msg::START_GAME)){
                execl("./gamer", "gamer", argv[1], "5555");
            }

            request = id + " " + input;
            response = socket.SendRequest(request);
        }

        if (response.length()){
            std::cout << response << '\n';
        }

    }

    request = cmd.TERMINATOR;
    socket.SendRequest(request);

    return 0;
}
```

server.cpp

```cpp
#include "utils.hpp"
#include "zmq_utils.hpp"
#include "allias.hpp"
#include "server_utils.hpp"

#include <cstdio>
#include <fstream>
#include <random>
#include <sstream>
#include <string>
#include <utility>
#include <vector>
#include <zmq.hpp>
#include <unistd.h>
#include <queue>
#include <set>

using pii = std::pair<int, int>;

class StatChecker
{
    private:

    public:
        std::map<std::string, int> data;
        StatChecker()
        {
            std::ifstream fin(room::PATH_TO_USERS_DB);

            std::string name;
            fin >> name;

            int record;
            fin >> record;

            data[name] = record;
        }
        bool Member(std::string &key){
            return data.count(key);
        }
        ~StatChecker()
        {
```

```
            std::ofstream fout(room::PATH_TO_USERS_DB);

            for (std::map<std::string, int>::iterator it = data.begin(); it !=
                fout << it->first << "␣" << it->second << '\n';
            }
        }
} stat;

struct TArgs
{
    int id;
    std::string command;
    std::string arg;
};

TArgs GetCommand(std::string &str)
{
    TArgs result;

    std::stringstream strm;
    strm << str;
    std::string str1;

    std::getline(strm, str1, '␣');
    result.id = std::atoi(str.data());
    std::getline(strm, str1, '␣');
    result.command = str1;

    if (!str1.compare(msg::KILL)){
        std::getline(strm, str1);
    }
    else {
        std::getline(strm, str1, '␣');
    }
    result.arg = str1;

    return result;
}

static std::set<std::string> logged;

std::string RunCommand(std::string &srcStr)
```

```
{
    std::string result;
    TArgs cmd = GetCommand(srcStr);

    RoomsManager manager;

    if (!cmd.command.compare(msg::INIT_CLIENT))
    {
        if (!logged.count(cmd.arg)){
            result = msg::NO_SUCH_USER;
        }
        else
        {
            logged.insert(cmd.arg);
            result = msg::SUCCES;
        }
    }

    if (!cmd.command.compare(msg::CREATE))
    {
        if (manager.Member(cmd.arg)){
            result = msg::ROOM_EXISTS;
        }
        else {
            result = manager.AddRoom(cmd.arg);
        }
    }

    if (!cmd.command.compare(msg::CONNECT))
    {
        if (!manager.Member(cmd.arg)){
            result = msg::ROOM_NOT_FOUND;
        }
        else {
            result = manager.GetPort(cmd.arg);
        }
    }

    if (!cmd.command.compare(msg::START_GAME))
    {
        int pid = fork();
```

```
        if (!pid){
            execl("./room", "room", "5555", NULL);
        }
    }

    if (!cmd.command.compare(msg::STAT))
    {
        if (!stat.Member(cmd.arg)){
            result = "0";
        }
        else
        {
            std::map<std::string, int>::iterator it = stat.data.find(cmd.arg)
            result = std::to_string(it->second);
        }
    }

    return result;
}

int main(int argc, char* argv[])
{
    std::string port = argv[1];
    std::string address = "tcp://*:" + port;
    zus::Socket socket(address, utl::SERVER);

    int numOfClients = 2;

    while (true)
    {
        std::string requestStr = socket.RecieveMessage();


        if (!requestStr.compare(utl::TERMINATOR))
        {
            --numOfClients;
            socket.SendMessage(requestStr);
            if (numOfClients < 1){
                break;
            }
        }
        else {
```

```
            socket.SendMessage(RunCommand(requestStr));
        }
    }

    return 0;
}
```

Выводы

Вызов *fork* дублирует породивший его процесс со всеми его переменными, файловыми дескрипторами, приоритетами процесса, рабочий и корневой каталоги, и сегментами выделенной памяти.

Ребёнок **не** наследует:

- идентификатора процесса (PID, PPID);
- израсходованного времени ЦП (оно обнуляется);
- сигналов процесса-родителя, требующих ответа;
- блокированных файлов (record locking).

В процессе выполнения лабораторной работы были приобретены навыки практического применения создания, обработки и отслеживания их состояния. Для выполнения данного варианта задания создание потоков как таковых не требуется, так как всю работу выполняет системный вызов «exec».